Yi Hu

HW 1

Github: https://github.com/imsashahu/6650-a1

Design

Classes

- Client.java: first, divideIntoThreads function in this class divide 500k requests into 183 threads. I choose to divide equally first and then add the remaining requests into these threads randomly. Second, executeInThreads function will initiate a CountDownLatch and execute all requests from every thread. When one thread finishes, CountDownLatch will count down 1. When CountDownLatch is count down to 0, we know that all threads are completed executed. Lastly, all printed job and writing performance data into CSV file will also be completed in this class.

- MyRunnable.java: a class that implements Runnable. In this class, run() is defined for the action that will be performed in each thread's all requests. In every thread, after all requests in this thread finish, it will tell the client that this thread is done, by counting down 1 from CountDownLatch.

- RandomGenerator.java: a class to generate all fields needed for one POST request, based on requirements. By calling generate(), fields for a complete POST request are refreshed with new random values.

Packages

- java.util.Random: based on suggestion of the HW instruction, this class is used to generate random numbers, given a certain range of integers. This class is mainly used in RandomGenerator.java.

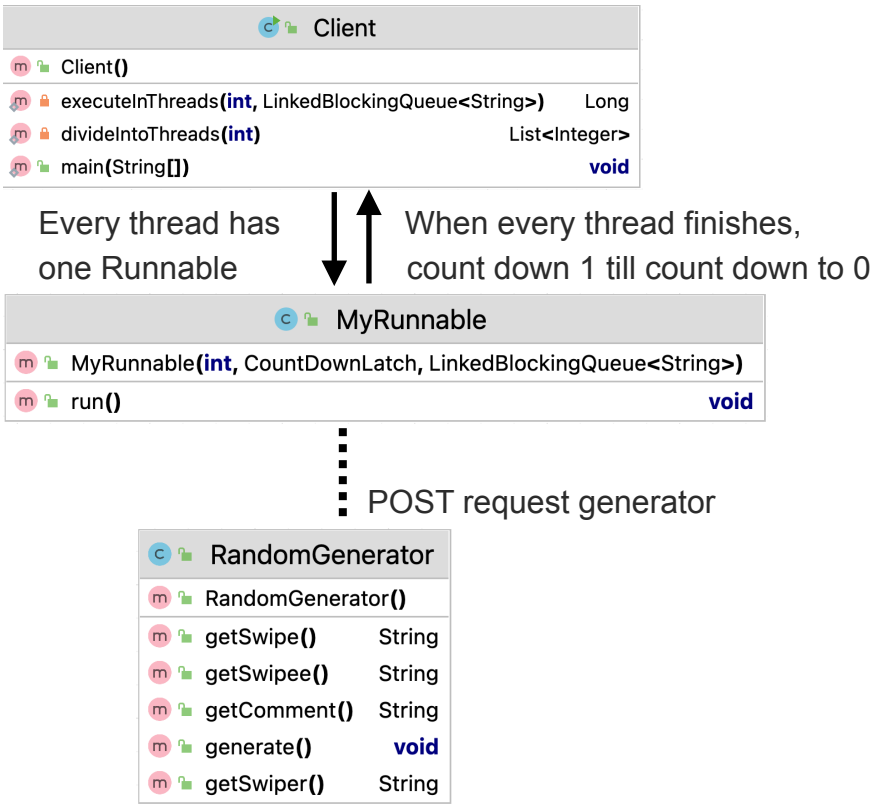
- java.sql.Timestamp: this is used for record start time and end time of the execution.

- java.util.concurrent.CountDownLatch: this is used to count down how many threads have been completed. Only when this count down is finished, we know that all threads and all requests have been completed successfully. This is synchronized.

- java.util.concurrent.LinkedBlockingQueue: this is a blocking queue that allows data to write into this queue one by one. This is thread safe, and so is used to record all the data that will be written into CSV file.

- java.io.FileWriter and java.io.BufferedWriter: use to write data in queue to a CSV file.

Relationships



Part 1 Result

```
/Users/SashaHu/Library/Java/JavaVirtualMachines/corretto-11.0.18/Contents/Home/bin/java ...
Total number of successful requests is: 500000
Total number of unsuccessful requests is: 0
Total run time in milliseconds is: 83150
Total throughput in requests per second is: 6013

Process finished with exit code 0
```

Number of threads: 183
Little's Law throughput predictions: 183 *(1000/29.288) = 6248.29

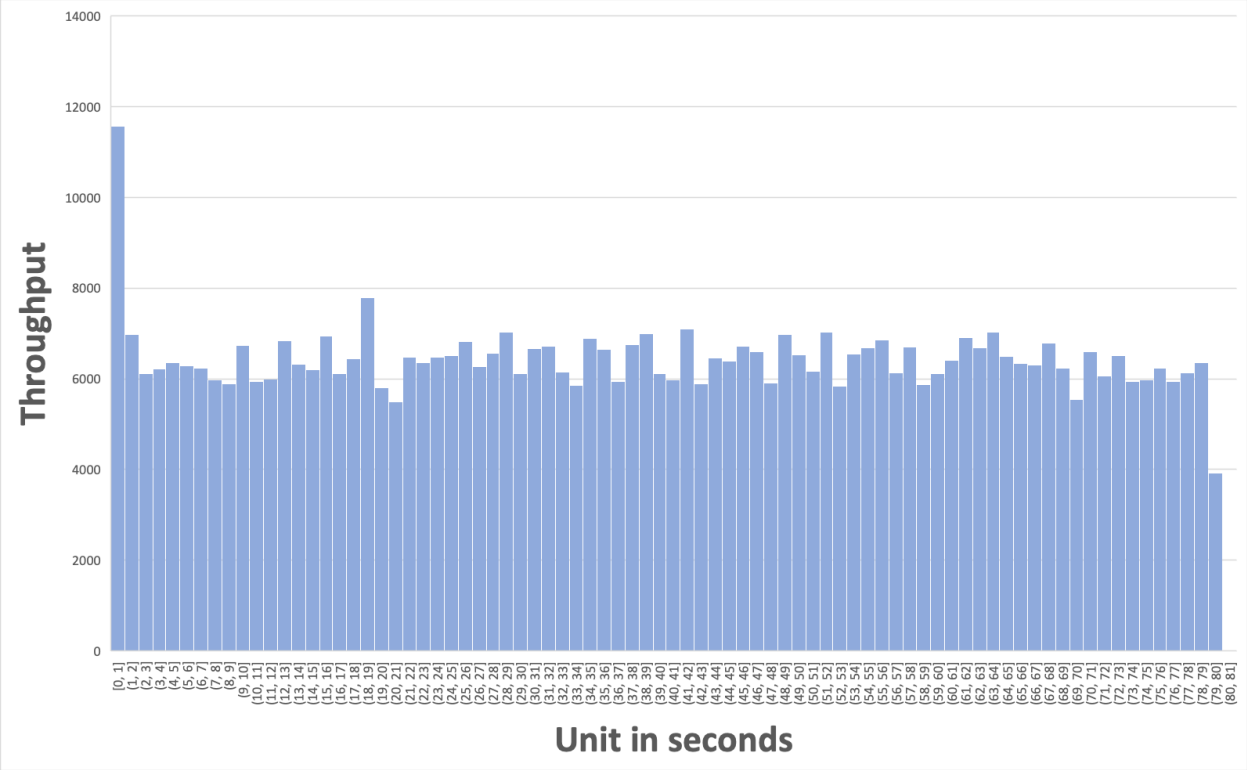
Part 2 Calculation

```
/Users/SashaHu/Library/Java/JavaVirtualMachines/corretto-11.0.18/Contents/Home/bin/java ...
Total number of successful requests is: 500000
Total number of unsuccessful requests is: 0
Total run time in milliseconds is: 81226
Total throughput in requests per second is: 6155

Process finished with exit code 0
```

- Mean response time (millisecs): 29.288
- Median response time (millisecs): 26
- Throughput requests/second: 6155
- 99th percentile response time (millisecs): 73
- Min response time (millisecs): 12
- Max response time (millisecs): 317

Plot Performance



Bonus

```
/Users/SashaHu/Library/Java/JavaVirtualMachines/corretto-11.0.18/Contents/Home/bin/java ...
Total number of successful requests is: 500000
Total number of unsuccessful requests is: 0
Total run time in milliseconds is: 201521
Total throughput in requests per second is: 2481

Process finished with exit code 0
```

Based on test results, the throughput of spring-boot server is much slower.